

## 1. Defina o que é Orientação a Objetos (OO)

Orientação a objetos é um paradigma de programação, isto é, uma forma de se pensar e de se estruturar um programa, com foco no uso de objetos para uma melhor organização, flexibilidade e manutenção dos códigos. Suas principais características são o encapsulamento, o polimorfismo, a herança e a mais fundamental: a abstração. A abstração torna entes (sejam eles explicitamente concretos ou abstratos) da vida real em objetos dentro do programa. Isto capacita a orientação a objetos a resolver problemas e necessidades reais do cotidiano.

## 2. Conceitos básicos

Explique os seguintes conceitos:

### a) Classe

Classes são como “tipos” de objetos. Elas possuem características e comportamentos gerais característicos dos objetos que as instanciam.

### b) Objeto

É uma instância de uma determinada classe, com atributos e métodos únicos (dentro do escopo permitido pela sua classe).

### c) Atributo

São qualidades (dados) de uma classe ou objeto, podem armazenar valores ou informações pertinentes.

### d) Método

São “ações” que um objeto pode fazer, basicamente são funções dentro do código que abstraem um comportamento de um objeto.

## 3. Conceito de classe e um objeto

Eu posso criar uma classe Cachorro, dar atributos a ela, e então instanciar um objeto, que é uma abstração de um cachorro da vida real, com os dados relativos a um objeto da classe Cachorro.

```
public class Cachorro { //tag: <!-- new -->
    // atributos da classe
    private static int qtdPatas; //tag: <!-- new -->
    private static float peso; //tag: <!-- new -->
    private static boolean estaDoente; //tag: <!-- new -->
    private static String cor; //tag: <!-- new -->

    //metodo construtor da classe
    Cachorro(int qtdPatas, float peso, boolean estaDoente, String cor) { //tag: <!-- new -->
        this.qtdPatas = qtdPatas;
        this.peso = peso;
        this.estaDoente = estaDoente;
        this.cor = cor;
    }

    // Objeto com os atributos da classe
    Cachorro principal = new Cachorro(4, peso: 45.5, estaDoente: 0, cor: "preta"); //tag: <!-- new -->
}
```

#### 4. Associação entre classes

Uma associação de classes ocorre quando uma classe se utiliza ou compõe outra. Por exemplo, classes Carro, e Pessoa podem fazer parte de uma classe Cliente em um sistema de uma oficina mecânica. Apar

#### 5. Criação de classe simples

```
© Apartamento.java x
1 public class Apartamento {
2     private static float area;
3     private static int quartos;
4     private static int andar;
5     private static float valorDeCompra;
6     private static int vagasDeGaragem;
7     private static boolean temVaranda;
8
9     public void exibirInfo(){
10        System.out.println(area);
11        System.out.println(quartos);
12        System.out.println(andar);
13        System.out.println(valorDeCompra);
14        System.out.println(vagasDeGaragem);
15        System.out.println(temVaranda);
16    }
17 }
```

#### 6. Herança

A herança consiste na relação Mãe-Filha entre classes. Uma classe filha herda atributos e métodos da mãe (e não o contrário).

```
© Veiculo.java x © Carro.java
1 package Main;
2
3 public class Veiculo { 1 inheritor
4     public static String cor;
5     public static float valor;
6 }
```

```
© Veiculo.java © Carro.java x
1 package Main;
2
3 public class Carro extends Veiculo {
4     private static int qtdAssentos;
5     private static boolean temCadeiraInfantil;
6     private float tamanhoVolante;
7
8     //Metodo construtor ja entende os atributos herdados de Veiculo
9
10    public Carro (String cor, float valor, int qtdAssentos, boolean temCadeiraInfantil, float tamanhoVolante){
11        this.cor = cor;
12        this.valor = valor;
13        this.qtdAssentos = qtdAssentos;
14        this.temCadeiraInfantil = temCadeiraInfantil;
15        this.tamanhoVolante = tamanhoVolante;
16    }
17 }
18
19 |
```

## 7. Polimorfismo - Sobrecarga

O polimorfismo é o principal artifício de adaptação e flexibilização de um programa orientado a objetos. Ele pode modificar ou mesmo reestruturar métodos de uma determinada classe, adequando-os as necessidades individuais de um objeto. A sobrecarga funciona pela alteração dos parâmetros de um método/operacão.

```
1 public class Matematica{
2     public static final int soma(int a, int b){
3         return a+b;
4     }
5
6     //Overload
7
8     public static final float soma(float a, float b){
9         return a+b;
10    }
11 }
```

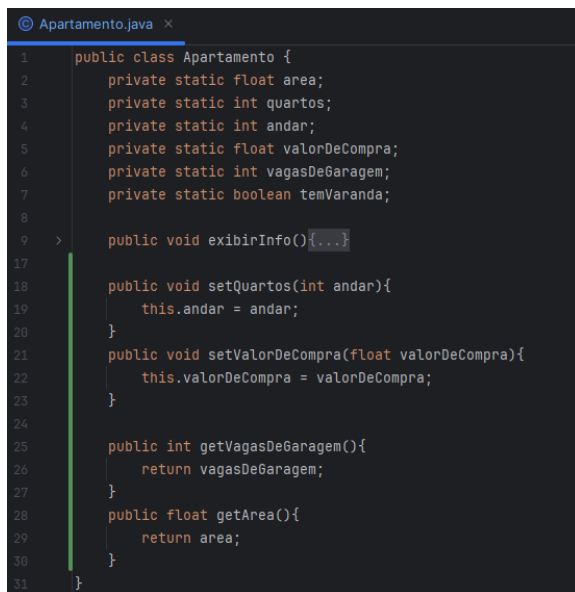
## 8. Polimorfismo - Sobrescrita

Agora, a assinatura do método permanece a mesma, isto é, seus parâmetros permanecem iguais. Porém, o método pode sofrer alterações em seus procedimentos. Além disso, a sobrescrita ocorre em classes filhas, ela se restringe a alterar um método herdado da classe mãe.

```
1 public class Calculadora { 1 inheritor
2     public float media(float n1, float n2, float n3) {
3         float media;
4         media = (n1 + n2 + n3) / 3;
5         return media;
6     }
7 }
8
9 public class CalculadoraAvancada extends Calculadora{
10
11     @Override
12     public float media(float n1, float n2, float n3){
13         float media;
14         media = (n1 + (n2 * 2) + (n3 * 3))/ 6;
15         return media;
16     }
17 }
```

## 9. Encapsulamento

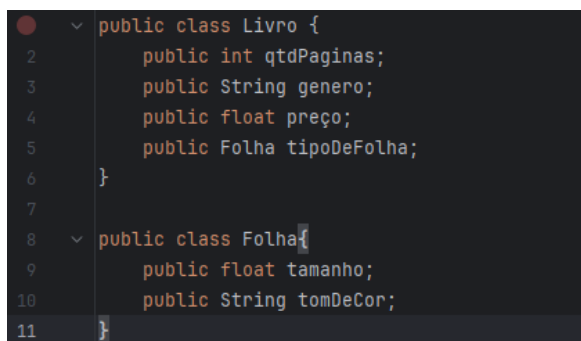
O encapsulamento se refere a estruturação e organização do código. Por exemplo, ele permite que determinados dados só possam ser acessados e/ou modificados por determinadas partes do programa. Além de compreender a coesão (o quão claro e objetivo é a relação entre classes e objetos) e o acoplamento (o nível de dependência e emanhamento entre classes).



```
1 public class Apartamento {
2     private static float area;
3     private static int quartos;
4     private static int andar;
5     private static float valorDeCompra;
6     private static int vagasDeGaragem;
7     private static boolean temVaranda;
8
9     public void exibirInfo(){...}
10
11     public void setQuartos(int andar){
12         this.andar = andar;
13     }
14     public void setValorDeCompra(float valorDeCompra){
15         this.valorDeCompra = valorDeCompra;
16     }
17
18     public int getVagasDeGaragem(){
19         return vagasDeGaragem;
20     }
21     public float getArea(){
22         return area;
23     }
24 }
```

## 10. Relacionamento de agregação

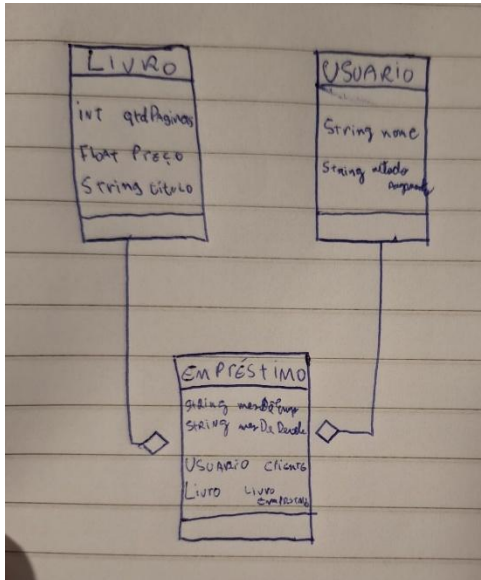
É uma associação fraca entre classes. Ocorre quando uma classe é instanciada contendo um objeto de outra classe como atributo, mas de modo em que existam objetos dessa outra classe de maneira independente.



```
1 public class Livro {
2     public int qtdPaginas;
3     public String genero;
4     public float preço;
5     public Folha tipoDeFolha;
6 }
7
8 public class Folha {
9     public float tamanho;
10    public String tomDeCor;
11 }
```

## 11. Projeto orientado a objetos

Desenhe um pequeno diagrama de classes para representar um sistema de gerenciamento de biblioteca, contendo pelo menos três classes (Livro, Usuario, Empréstimo) e depois implemente essas classes em Java.



```
1  public class Livro{
2      private int qtdPaginas;
3      private float preço;
4      private String titulo;
5  }
6
7  public class Usuario{
8      private String nome;
9      private String metodoPagamento;
10 }
11
12 public class Empréstimo{
13     private String mesDoEmpréstimo;
14     private String mesDaDevolução;
15     private Usuario cliente;
16     private Livro livroEmprestado;
17 }
```

## 12. Projeto orientado a objetos 2

```
1 package Classes;
2
3 public class Livro{
4     private int qtdPaginas;
5     public String titulo;
6
7     public Livro(String titulo, int qtdPaginas){
8         this.qtdPaginas = qtdPaginas;
9         this.titulo = titulo;
10    }
11
12    public String getTitulo(){
13        return titulo;
14    }
15 }
16
```

```
1 package Classes;
2
3 public class Usuario {
4     public String nome;
5     private int idade;
6
7     public Usuario(String nome, int idade){
8         this.nome = nome;
9         this.idade = idade;
10    }
11
12    public String getNome(){
13        return nome;
14    }
15 }
16
```

```
1 package Classes;
2
3 import java.time.LocalDate;
4
5 public class Empréstimo {
6     private LocalDate dataEmpréstimo;
7     private LocalDate dataDevolução;
8     public Livro livro;
9     public Usuario cliente;
10
11
12     public Empréstimo(Usuario cliente, Livro livro, LocalDate dataEmpréstimo, LocalDate dataDevolução){
13         this.livro = livro;
14         this.cliente = cliente;
15         this.dataEmpréstimo = dataEmpréstimo;
16         this.dataDevolução = dataDevolução;
17     }
18
19     public void doEmpréstimo() {
20         System.out.println("Empréstimo concluído!");
21         System.out.println("Livro: " + livro.getTitulo());
22         System.out.println("Usuário: " + cliente.getNome());
23         System.out.println("Data de Empréstimo: " + dataEmpréstimo);
24     }
25 }
```

```
public void doDevolucao() {  
    System.out.println("Devolução concluída!");  
    System.out.println("Livro: " + livro.getTitulo());  
    System.out.println("Usuário: " + cliente.getNome());  
    System.out.println("Data de Devolução: " + dataDevolução);  
}
```

```
Main.java x Livro.java Usuario.java Emprestimo.java  
1 import Classes.*;  
2  
3 import java.time.LocalDate;  
4 import java.util.Scanner;  
5  
6 public class Main{  
7     public static void main(String[] args) {  
8         Livro livro = new Livro(titulo: "Harry Potter", qtdPaginas: 350);  
9         Usuario usuario = new Usuario(nome: "Daniel", idade: 18);  
10  
11         Emprestimo emprestimo = new Emprestimo(usuario, livro,  
12             LocalDate.of(year: 2025, month: 9, dayOfMonth: 11), LocalDate.of(year: 2025, month: 10, dayOfMonth: 11));  
13  
14  
15         emprestimo.doEmprestimo();  
16         emprestimo.doDevolucao();  
17     }  
18 }  
19 |
```